Linux Kernel Networking- advanced topics: Neighboring and IPsec

Rami Rosen
ramirose@gmail.com
Haifux, January 2008
www.haifux.org

Contents

Scope

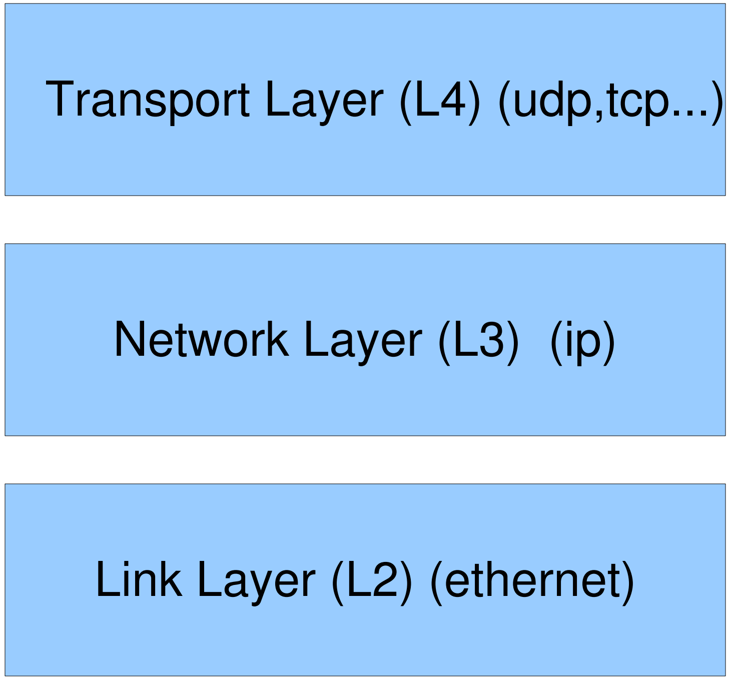
- We will not deal with multicast and with ipv6 and with wireless.
- The L3 network protocol we deal with is ipv4, and the
  L2 Link Layer protocol is Ethernet.

Neighboring Subsystem

- All code in this lecture is taken from linux-2.6.24-rc4
- 04-Dec-2007
- Can be obtained from http://www.kernel.org/pub/linux/kernel/v2.6/testing/ (and mirrors)

Short rehearsal (4 slides)

- **The layers that we will deal with (based on the 7 layers model) are:**



Short rehearsal (4 slides)

- Two most Important data structures: sk_buff and net_device.

sk_buff:

- dst is an instance of dst_entry; dst is a member in sk_buff.
- The lookup in the routing subsystem constructs dst.
- It decides how the packet will continue its traversal.
- This is done by assigning methods to its input()/output() functions
- Each dst_entry has a neighbour member.(with IPSec it is NULL).
- When working with IPSec, the dst in fact represents a linked list of dst_entries. Only the last one is for routing; all previous dst_entries are for IPSec transformers.

Short rehearsal (4 slides)

net_device

- net_device represents a Network Interface Card.
- net_device has members like mtu, dev_addr (device MAC address), promiscuity,name of device (eth0,eth1,lo, etc), and more.
- An important member of net_device is flags.
- You can disable ARP replies on a NIC by setting IFF_NOARP flag:
- ***ifconfig eth0 -arp***
  - ***ifconfig eth0*** will show:
    - UP BROADCAST RUNNING NOARP MULTICAST ...
  - Enabling ARP again is done by: ***ifconfig eth0 arp.***

Short rehearsal (4 slides)

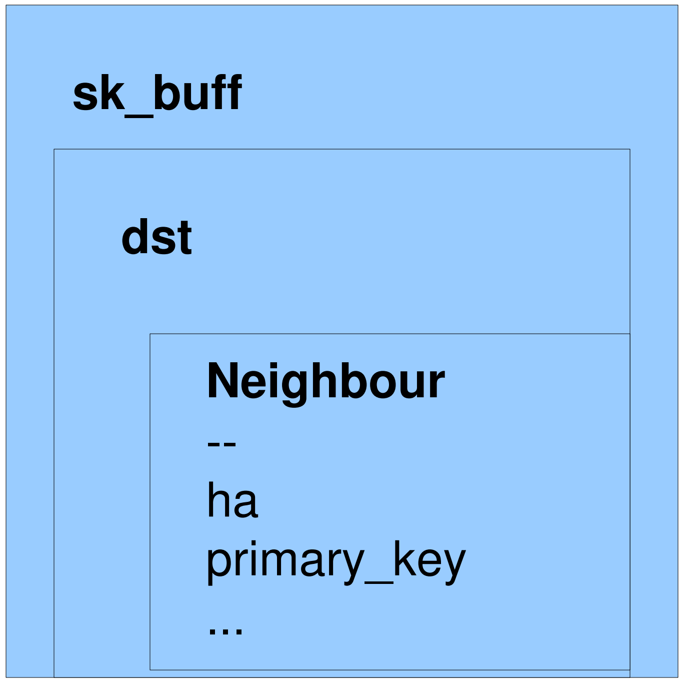
- ***ip_input_route()*** method: performs a lookup in the routing subsystem for each incoming packet. Looks first in the routing cache; in case there is a cache miss, looks into the routing table and inserts an entry into the routing cache. Calls arp_bind_neighbour() for UNICAST packets only. Returns 0 upon success.
- ***dev_queue_xmit(struct sk_buff *skb)*** is called to transmit the packet, when it is ready. (has L2 destination address) *(net/core/dev.c)*
  - ***dev_queue_xmit()*** passes the packet to the nic device driver for transmission using the device driver ***hard_start_xmit()*** method.

Neighboring Subsystem

- **Goals: what is the neighboring subsystem for?**
- *"The world is a jungle in general, and the networking game contributes many animals."* (from RFC 826, ARP, 1982)
- In IPV4 implemented by **ARP**; in IPv6: **ND**, neighbour discovery.
- Ethernet header is 14 bytes long:
- Source Mac address and destination Mac address - 6 bytes each.
  - Type (2 bytes). For example, (*include/linux/if_ether.h)*
    - 0x0800 is the type for IP packet **(ETH_P_IP)**
    - 0x0806 is the type for ARP packet **(ETH_P_ARP)**
    - 0X8035 is the type for RARP packet **(ETH_P_RARP)**

Neighboring Subsystem – struct neighbour

- neighbour (instance of struct neighbour) is embedded in dst, which is in turn is embedded in sk_buff:



Neighboring Subsystem – struct neighbour

- Implementation - important data structures
- struct neighbour *(/include/et/neighbour.h)*
  - *ha* - the hardware address (MAC address when dealing with Ethernet) of the neighbour. This field is filled when an ARP response arrives.
  - *primary_key* – The IP address (L3) of the neighbour.
    - lookup in the arp table is done with the primary_key.
  - *nud_state* represents the Network Unreachability Detection

    state of the neighbor. (for example, NUD_REACHABLE).

Neighboring Subsystem – struct neighbour contd

- A neighbour can change its state to NUD_REACHABLE by one of three ways:
- L4 confirmation.
- Receive ARP reply for the first time or receiving an ARP reply in response to an ARP request when in NUD_PROBE state.
- Confirmation can be done also by issuing a sysadmin command (but it is rare).

Neighboring Subsystem – struct neighbour contd

- *int (*output)(struct sk_buff *skb);*
  - *output()* can be assigned to different methods according to the state of the neighbour. For example, ***neigh_resolve_output()*** and ***neigh_connected_output()***. Initially, it is ***neigh_blackhole()*.**
  - When a state changes, than also the output function may be assigned to a different function.
- *refcnt* -incremented by *neigh_hold()*; decremented by

  neigh_release(). We don't free a neighbour when the refcnt is higher than 1; instead, we set dead (a member of neighbour) to 1.

Neighboring Subsystem – struct neighbour contd

- timer (The callback method is *neigh_timer_handler()*).
- *struct hh_cache *hh (defined in include/linux/netdevice.h)*
- *confirmed –* confirmation timestamp.
  - Confirmation can done from L4 (transport layer).
  - For example, ***dst_confirm()*** calls ***neigh_confirm().***
  - ***dst_confirm()*** is called from ***tcp_ack()*** (*net/ipv4/tcp_input.c*)
  - and by ***udp_sendmsg()*** (net/ipv4/udp.c) and more.
  - ***neigh_confirm()* does NOT change the state – it is the job of *neigh_timer_handler().***

Neighboring Subsystem – struct neighbour contd

- *dev* (net_device from which we send packets to the neighbour).
- *struct neigh_parms   *parms;*
  - parms include mostly timer tunables, net structure (network namespaces), etc.
  - network namespaces enable multiple instances of the network stack to the user space.
- A network device belongs to exactly one network namespace.
- CONFIG_NET_NS when building the kernel.

Neighboring Subsystem – struct neighbour contd

- *arp_queue*
  - every neighbour has a small arp queue of itself.
  - There can be only 3 elements by default in an arp_queue.
  - This is configurable:*/proc/sys/net/ipv4/neigh/default/unres_qlen*

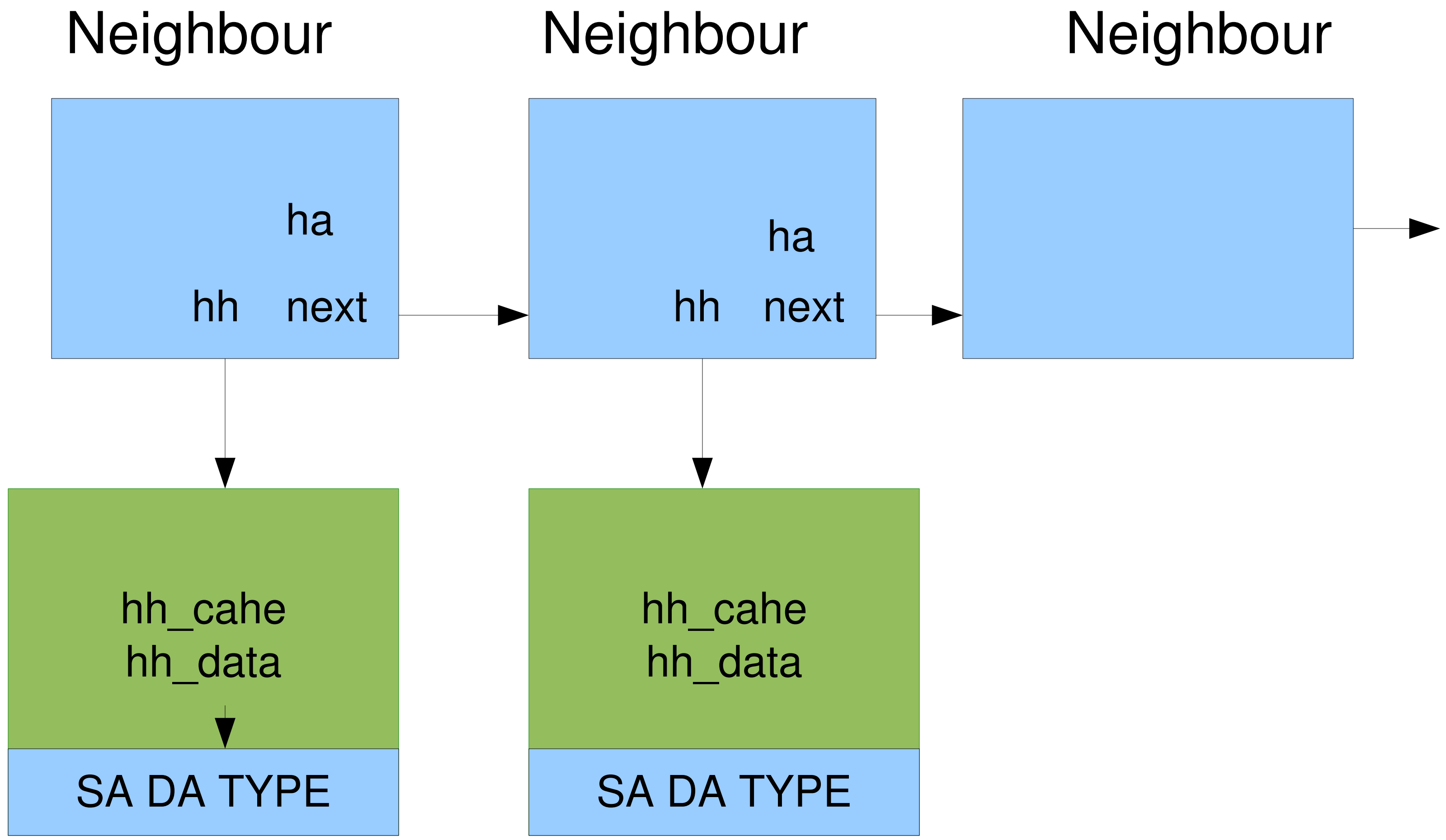
struct neigh_table

- struct neigh_table represents a neighboring table
  - *(/include/net/neighbour.h)*
  - The **arp table** (arp_tbl) is a neigh_table. *(/include/net/arp.h)*
  - In IPv6, **nd_tbl** (Neighbor Discovery table) is a neigh_table also *(include/net/ndisc.h)*
  - There is also **dn_neigh_table** (DECnet) (*linux/net/decnet/dn_neigh.c)* and **clip_tbl** (for ATM) (*net/atm/clip.c)*
  - *gc_timer : neigh_periodic_timer()* is the callback for garbage collection.
  - *neigh_periodic_timer()* deletes FAILED entries from the ARP table.

Neighboring Subsystem - arp

- When there is no entry in the ARP cache for the destination IP address of a packet, a broadcast is sent (ARP request, `ARPOP_REQUEST:` who has IP address x.y.z...). This is done by a method called ***arp_solicit()***. (*net/ipv4/arp.c*)
  - In IPv6, the parallel mechanism is called ND (Neighbor discovery) and is implemented as part of ICMPv6.
  - A multicast is sent in IPv6 (and not a broadcast).
- If there is no answer in time to this arp request, then we will end up with sending back an ICMP error (Destination Host Unreachable).
- This is done by ***arp_error_report()*** , which indirectly calls ***ipv4_link_failure()*** ; see net/ipv4/route.c.

ARP table


Neighbour
Neighbour
Neighbour
ha
hh
next
ha
hh
next
hh_cahe
hh_data
SA DA TYPE
hh_cahe
hh_data
SA DA TYPE

Neighboring Subsystem - arp

- You can see the contents of the arp table by running:

"*cat /proc/net/arp*" or by running the "arp" from a command line .

- *ip neigh show* is the new method to show arp (from IPROUTE2)
- You can delete and add entries to the arp table; see man arp/man ip.
- When using "ip neigh add" you can specify the state of the entry which you are adding (like permanent,stale,reachable, etc).

Neighboring Subsystem – arp table

- arp command does not show reachability states except the incomplete state and permanent state:

Permanent entries are marked with M in Flags:

example : arp output

```
Address          HWtype  HWaddress          Flags Mask     Iface
10.0.0.2                              (incomplete)         eth0
10.0.0.3          ether  00:01:02:03:04:05   CM             eth0
10.0.0.138        ether  00:20:8F:0C:68:03   C              eth0
```

Neighboring Subsystem – ip show neigh

- We can see the current neighbour states:
- Example :
- ***ip neigh show***

```
192.168.0.254 dev eth0 lladdr 00:03:27:f1:a1:31 REACHABLE
192.168.0.152 dev eth0 lladdr 00:00:00:cc:bb:aa STALE
192.168.0.121 dev eth0 lladdr 00:10:18:1b:1c:14 PERMANENT
192.168.0.54   dev eth0 lladdr aa:ab:ac:ad:ae:af  STALE
192.168.0.98 dev eth0                              INCOMPLETE
```

Neighboring Subsystem – arp

- *arp_process()* handles both ARP requests and ARP responses.
  - *net/ipv4/arp.c*
  - If the target ip (tip) address in the arp header is the loopback then *arp_process()* drops it since loopback does not need ARP.

```
...
if (LOOPBACK(tip) || MULTICAST(tip))
  goto out;
out:
...
  kfree_skb(skb);
  return 0;
```

Neighboring Subsystem - arp

(see: #define LOOPBACK(x) (((x) & htonl(0xff000000)) == htonl(0x7f000000)) in linux/in.h

- If it is an ARP request (ARPOP_REQUEST)

  we call *ip_route_input().*
- Why ?
- In case it is for us, (RTN_LOCAL) we send and ARP reply.
  - *arp_send(ARPOP_REPLY,ETH_P_ARP,sip,dev,tip,sha*

    ,dev->dev_addr,sha);
  - We also update our arp table with the sender entry (ip/mac).
- Special case: ARP proxy server.

Neighboring Subsystem - arp

- In case we receive an **ARP reply – (ARPOP_REPLY)**
  - We perform a lookup in the arp table. (by calling *__neigh_lookup()*)
  - If we find an entry, we update the arp table by

    neigh_update().

Neighboring Subsystem - arp

- If there is no entry and there is NO support for unsolicited ARP we don't create an entry in the arp table.
  - Support for unsolicited ARP by setting /proc/sys/net/ipv4/conf/all/arp_accept to 1.
  - The corresponding macro is: *IPV4_DEVCONF_ALL(ARP_ACCEPT))*
  - In older kernels, support for unsolicited ARP was done by:
  - *CONFIG_IP_ACCEPT_UNSOLICITED_ARP*

Neighboring Subsystem – lookup

- Lookup in the neighboring subsystem is done via: *neigh_lookup()*

parameters:

  - neigh_table (arp_tbl)
  - pkey (ip address, the primary_key of neighbour struct)
  - dev (net_device)
  - There are 2 wrappers:
  - *__neigh_lookup()*
    - just one more parameter: creat (a flag: to create a neighbor by neigh_create() or not))

- and *__neigh_lookup_errno()*

Neighboring Subsystem – static entries

- Adding a static entry is done by arp -s ipAddress MacAddress
- Alternatively, this can be done by:

ip neigh add ipAddress dev eth0 lladdr MacAddress nud permanent

- The state (*nud_state*) of this entry will be NUD_PERMANENT
  - ip neigh show will show it as PERMANENT.
- Why do we need PERMANENT entries ?

arp_bind_neighbour() method

- Suppose we are sending a packet to a host for the first time.
- a dst_entry is added to the routing cache by *rt_intern_hash()*.
- We should know the L2 address of that host.
  - so *rt_intern_hash()* calls *arp_bind_neighbour().*
    - **only** for RTN_UNICAST (not for multicast/broadcast).
  - *arp_bind_neighbour()*: *net/ipv4/arp.c*
  - dst->neighbour=NULL, so it calls*__neigh_lookup_errno()*.
  - There is no such entry in the arp table.
  - So we will create a neighbour with *neigh_create()* and add it to the arp table.

arp_bind_neighbour() method

- *neigh_create()* creates a neighbour with NUD_NONE state
  - setting nud_state to NUD_NONE is done in neigh_alloc()

Neighboring Subsystem – IFF_NOARP flag

- Disabling and enabling arp
- ifconfig eth1 -arp
  - You will see the NOARP flag now in ifconfig -a
- ifconfig eth1 arp (to enable arp of the device).
- In fact, this sets the IFF_NOARP flag of net_device.
- There are cases where the interface by default is with the IFF_NOARP flag (for example, ppp interface, see *ppp_setup()* (*drivers/net/ppp_generic.c)*

Changing IP address

- Suppose we try to set eth1 to an IP address of a different machine on the LAN:
- First, we will set an ip for eth1 in (in FC8,for example)
- /etc/sysconfig/network-scripts/ifcfg-eth1

```
...
IPADDR=192.168.0.122
...
```

and than run:

- ifup eth1

Changing IP address - contd.

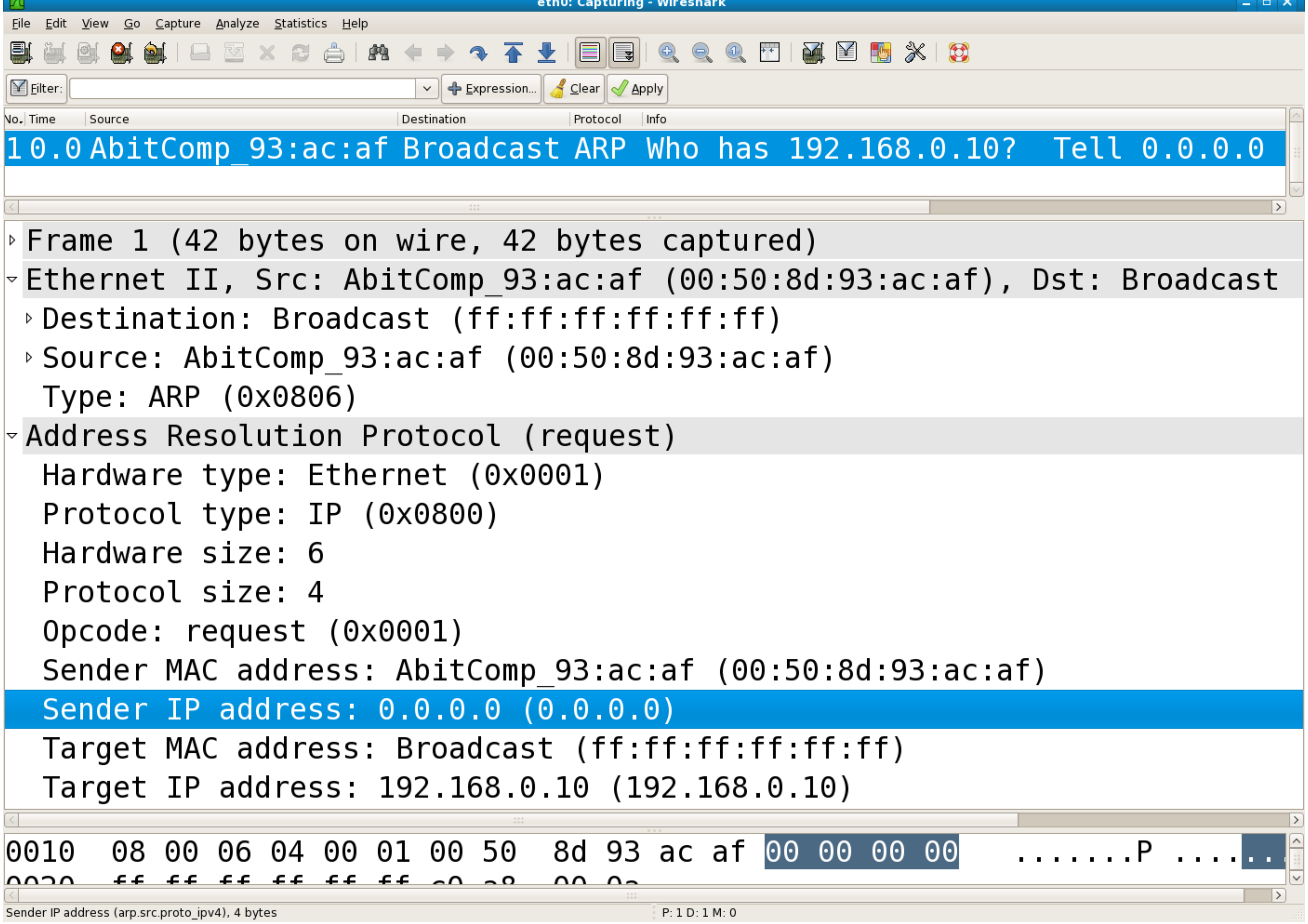
- we will get:

```
Error, some other host already uses address
192.168.0.122.
```

- But:
- `ifconfig eth0 192.168.0.122`
- works ok !
- Why is it so ?
- ifup is from the initscripts package.

Duplicate Address Detection (DAD)

- Duplicate Address Detection mode (DAD)
- arping -I eth0 -D 192.168.0.10
  - sends a broadcast packet whose source address is 0.0.0.0.
- 0.0.0.0 is not a valid IP address (for example, you cannot set an ip address to 0.0.0.0 with ifconfig)
- The mac address of the sender is the real one.
- -D flag is for Duplicate Address Detection mode.


eth0: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. Time Source               Destination  Protocol Info
1   0.0  AbitComp_93:ac:af    Broadcast    ARP      Who has 192.168.0.10?  Tell 0.0.0.0
```

```
▷ Frame 1 (42 bytes on wire, 42 bytes captured)
▽ Ethernet II, Src: AbitComp_93:ac:af (00:50:8d:93:ac:af), Dst: Broadcast
  ▷ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▷ Source: AbitComp_93:ac:af (00:50:8d:93:ac:af)
    Type: ARP (0x0806)
▽ Address Resolution Protocol (request)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: AbitComp_93:ac:af (00:50:8d:93:ac:af)
    Sender IP address: 0.0.0.0 (0.0.0.0)
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 192.168.0.10 (192.168.0.10)
```

```
0010  08 00 06 04 00 01 00 50  8d 93 ac af 00 00 00 00   .......P ........
0020  ff ff ff ff ff ff c0 a8  00 0a
```

Sender IP address (arp.src.proto_ipv4), 4 bytes   P: 1 D: 1 M: 0

Duplicate Address Detection -contd

Code: (from *arp_process()* ; see /net/ipv4/arp.c)

```
/* Special case: IPv4 duplicate address detection packet (RFC2131)
    */
if (sip == 0) {
if (arp->ar_op == htons(ARPOP_REQUEST) &&
    inet_addr_type(tip) == RTN_LOCAL &&
    !arp_ignore(in_dev,dev,sip,tip))
arp_send(ARPOP_REPLY,ETH_P_ARP,tip,dev,tip,sha,dev-
    >dev_addr,dev->dev_addr);
goto out;
}
```

Neighboring Subsystem – Garbage Collection

- Garbage Collection
  - *neigh_periodic_timer()*
  - *neigh_timer_handler()*
  - *neigh_periodic_timer()* removes entires which are in NUD_FAILED state. This is done by setting dead to 1, and calling ***neigh_release()***. The refcnt must be 1 to ensure no one else uses this neighbour. Also expired entries are removed.
- **NUD_FAILED** entries don't have MAC address ; see “ip neigh show” in the example above).

Neighboring Subsystem – Asynchronous Garbage Collection

- *neigh_forced_gc()* performs synchronous garbage collection.
- It is called from *neigh_alloc()* when the number of the entries in the arp table exceeds a (configurable) limit.
- This limit is configurable (gc_thresh2,gc_thresh3)

  /proc/sys/net/ipv4/neigh/default/gc_thresh2

  /proc/sys/net/ipv4/neigh/default/gc_thresh3

  – The default for gc_thresh3 is 1024.
  – Candidates for cleanup: Entries which their reference count is 1, or which their state is NOT permanent.

Neighboring Subsystem – Garbage Collection

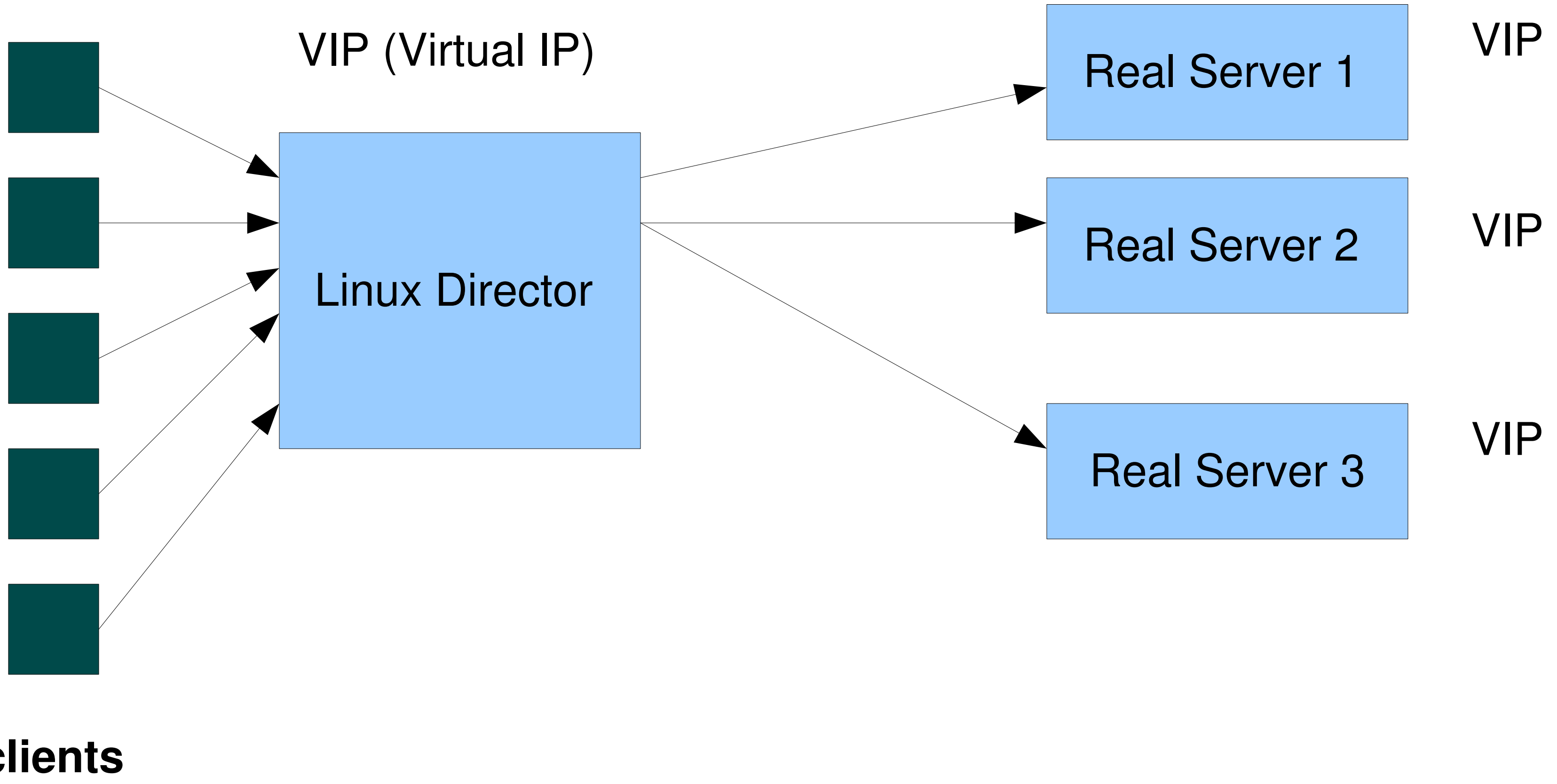
- Changing the neighbour state is done only in *neigh_timer_handler()* .

LVS (Linux Virtual Sever)

- http://www.linuxvirtualserver.org/
- Integrated into the Linux kernel (in 2.4 kernel it was a patch).
- Located in: *net/ipv4/ipvs* in the kernel tree. No IPV6 support.
- LVS has eight scheduling algorithms.
- LVS/DR is LVS with direct routing (a load balancing solution).
- ipvsadm is the user space management tools (available in most distros).
- Direct Routing is the packet-forwarding-method.
    - -g, --gatewaying => Use gatewaying (direct routing)
    - see man ipvsadm.

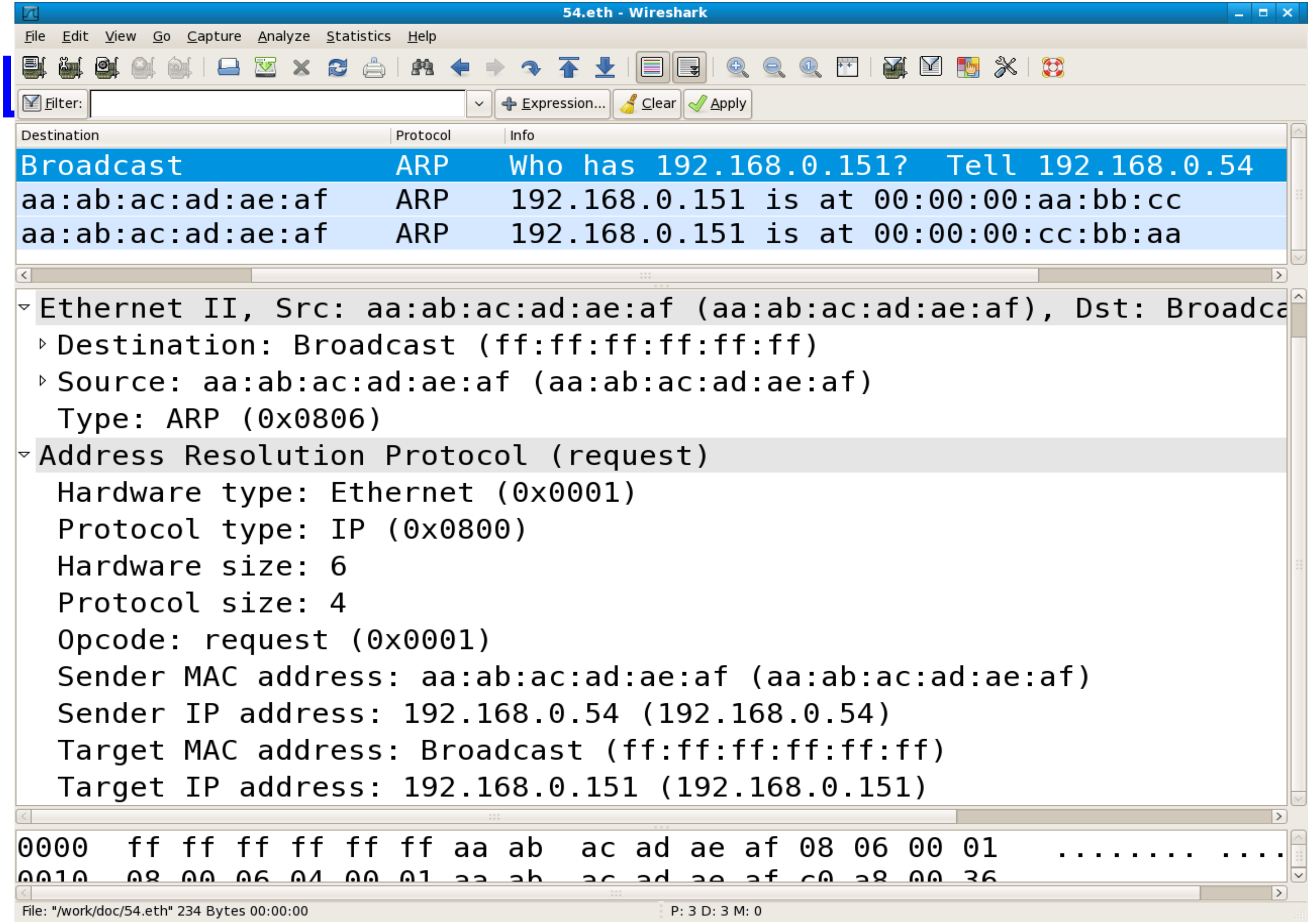
LVS/DR

- Example: 3 **Real Servers** and the **Director** all have the same Virtual IP (VIP).



LVS and ARP

- There is an ARP problem in this configuration.
- When you send an ARP broadcast, and the receiving machine has two or more NICs, each of them responds to this ARP request.
-
- Example: a machine with two NICs ;
- eth0 is 192.168.0.151 and eth1 is 192.168.0.152.



```
Destination          Protocol  Info
Broadcast            ARP       Who has 192.168.0.151?  Tell 192.168.0.54
aa:ab:ac:ad:ae:af    ARP       192.168.0.151 is at 00:00:00:aa:bb:cc
aa:ab:ac:ad:ae:af    ARP       192.168.0.151 is at 00:00:00:cc:bb:aa

▽ Ethernet II, Src: aa:ab:ac:ad:ae:af (aa:ab:ac:ad:ae:af), Dst: Broadca
  ▷ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▷ Source: aa:ab:ac:ad:ae:af (aa:ab:ac:ad:ae:af)
    Type: ARP (0x0806)
▽ Address Resolution Protocol (request)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: aa:ab:ac:ad:ae:af (aa:ab:ac:ad:ae:af)
    Sender IP address: 192.168.0.54 (192.168.0.54)
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 192.168.0.151 (192.168.0.151)

0000  ff ff ff ff ff ff aa ab  ac ad ae af 08 06 00 01   ........ ....
0010  08 00 06 04 00 01 aa ab  ac ad ae af c0 a8 00 36

File: "/work/doc/54.eth" 234 Bytes 00:00:00          P: 3 D: 3 M: 0
```

LVS and ARP

- Solutions

1) Set ARP_IGNORE to 1:
   - *echo "1" > /proc/sys/net/ipv4/conf/eth0/arp_ignore*
   - *echo "1" > /proc/sys/net/ipv4/conf/eth1/arp_ignore*

2) Use arptables.
   - There are 3 points in the arp walkthrough: (include/linux/netfilter_arp.h)
   - NF_ARP_IN (in *arp_rcv()* , *net/ipv4/arp.c*).
   - NF_ARP_OUT (in *arp_xmit()*),*net/ipv4/arp.c)*
   - NF_ARP_FORWARD (in *br_nf_forward_arp()*, *net/bridge/br_netfilter.c*)

LVS and ARP

- http://ebtables.sourceforge.net/download.html
  - Ebtables is in fact the parallel of netfilter but in L2.

LVS example (ipvsadm)

- An example for setting LVS/DR on TCP port 80 with three real servers:
- **ipvsadm -C** // clear the LVS table
- **ipvsadm -A -t DirectorIPAddress:80**
- **ipvsadm -a -t DirectorIPAddress:80 -r RealServer1 -g**
- **ipvsadm -a -t DirectorIPAddress:80 -r RealServer2 -g**
- **ipvsadm -a -t DirectorIPAddress:80 -r RealServer3 -g**
- This example deals with tcp connections (for udp connection we should use -u instead of -t in the last 3 lines).

LVS example:

- **ipvsadm -Ln** // list the LVS table
- **/proc/sys/net/ipv4/ip_forward** should be set to 1
- In this example, packets sent to VIP will be sent to the load balancer; it will delegate them to the real server according to its scheduler. The dest MAC address in L2 header will be the MAC address of the real server to which the packet will be sent. The dest IP header will be VIP.
- This is done with NF_IP_LOCAL_IN.

ARPD – arp user space daemon

- ARPD is a user space daemon; it can be used if we want to remove some work from the kernel.
- The user space daemon is part of iproute2 *(/misc/arpd.c)*
- **ARPD has support for negative entries and for dead hosts.**
  - **The kernel arp code does NOT support these type of entries!**
- The kernel by default is not compiled with ARPD support; we should set CONFIG_ARPD for using it:
- Networking Support->Networking Options->IP: ARP daemon support. (It is considered “Experimental”).
- see: /usr/share/doc/iproute-2.6.22/arpd.ps (Alexey Kuznetsov).

ARPD

- We should also set app_probes to a value greater than 0 by setting
  - */proc/sys/net/ipv4/neigh/eth0/app_solicit*
  - This can be done also by the -a (active_probes) parameter.
  - The value of this parameter tells how many ARP requests to send before that neighbour is considered dead.
- The -k parameter tells the kernel not to send ARP broadcast; in such case, the arpd daemon is not only listening to ARP requests, but also send ARP broadcasts.
- We can tune kernel parameters as we like; in fact, we can tune it so that arp requests will be send only from the daemon and not from the kernel at all.

ARPD

- Activation:
- *arpd -a 1 -k eth0 &*
- On some distros, you will get the error *db_open: No such file or directory* unless you simply run mkdir /var/lib/arpd/ before (for the arpd.db file).
- Pay attention: you can start arpd daemon when there is no support in the kernel (CONFIG_ARPD is not set).
- In this case you, arp packets are still caught by arpd daemon *get_arp_pkt()* (misc/arpd.c)
- But you don't get messages from the kernel.
- *get_arp_pkt()* is not called. (misc/arpd.c)

ARPD

- Tip: to check if CONFIG_ARPD is set, simply see if there are any resulrs from
  - ***cat /proc/kallsyms | grep neigh_app***

Mac addresses

- MAC address (Media Access Control)
- According to specs, MAC address should be unique.
- The 3 first bytes specify a hw manufacturer of the card.
- Allocated by IANA.
  - There are exceptions to this rule.
  - Technion (?)
  - Ethernet HWaddr 00:16:3E:3F:6E:5D

ARPwatch (detect ARP cache poisoning)

- Changing MAC address can be as a result of some security attack (ARP cache poisoning, ARP spoofing).
- **Arpwatch** is an open source tool;helps to detect such attack.
- Activation: arpwatch -d -i eth0 (output to stderr)
- Arpwatch keeps a table of ip/mac addresses and senses when there is a change.
- -d is for redirecting the log to stderr (no syslog, no mail).
- In case someone changed MAC address on the same network, you will get a message like this:

ARPwatch - Example

```
From: root (Arpwatch)
To: root
Subject: changed ethernet address (jupiter)
          hostname: jupiter
          ip address: 192.168.0.54
          ethernet address: aa:bb:cc:dd:ee:ff
          ethernet vendor: <unknown>
          old ethernet address: 0:20:18:61:e5:e0
          old ethernet vendor: ...
```

Change of IP address/Mac address

- Change of IP address does not trigger notifying its neighbours.
- Change of MAC address , **NETDEV_CHANGEADDR**, also does not trigger notifying its neighbours.
- It does update the local arp table by *neigh_changeaddr().*
  - Exception to this is irlan eth: *irlan_eth_send_gratuitous_arp()*
  - (*net/irda/irlan/irlan_eth.c*)
  - Some nics don't permit changing of MAC address – you get: SIOCSIFHWADDR: Device or resource busy
  - Sometimes you should only bring down the nic before.

Flushing the arp table

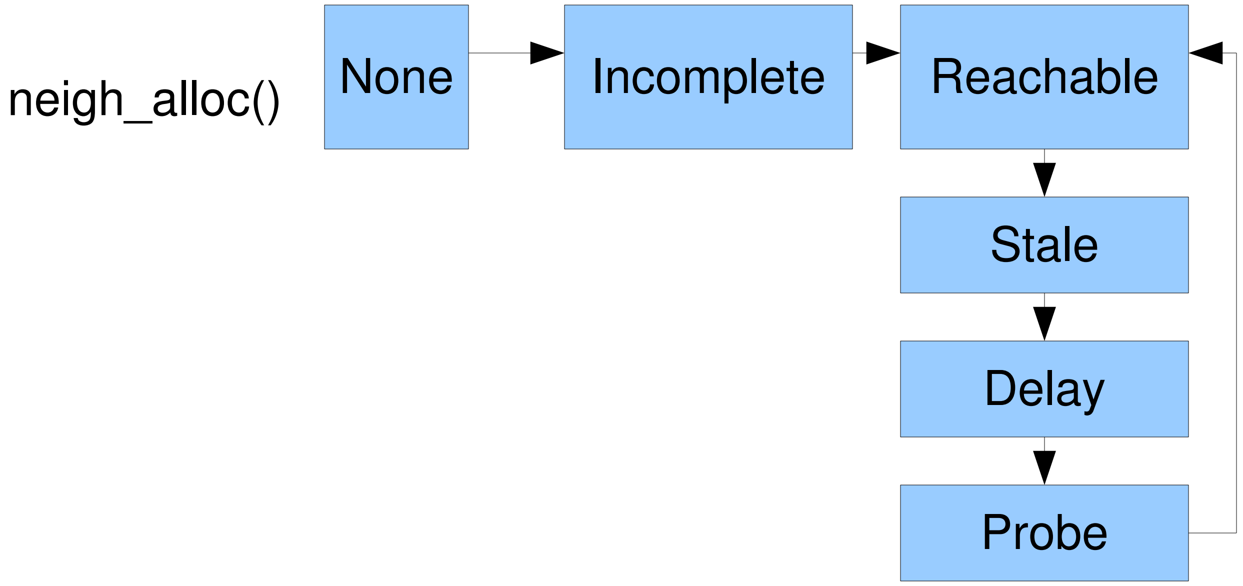
- Flushing the arp:
- ip -statistics neigh flush dev eth0
- *** Round 1, deleting 7 entries ***
- *** Flush is complete after 1 round ***

Flushing the arp table -contd

- Specifying twice -statistics will also show which entries were deleted, their mac addresses, etc...
- ip -statistics -statistics neigh flush dev eth0
- 192.168.0.254 lladdr 00:04:27:fd:ad:30 ref 17 used 0/0/0 REACHABLE
-
- *** Round 1, deleting 1 entries ***
- *** Flush is complete after 1 round ***
- calls neigh_delete() in net/core/neighbour.c
- Changes the state to NUD_FAILED

Neighbour states

- neighbour states



Neighboring Subsystem – states

- NUD states
  - NUD_NONE
  - NUD_REACHABLE
  - NUD_STALE
  - NUD_DELAY
  - NUD_PROBE
  - NUD_FAILED
  - NUD_INCOMPLETE

Neighboring Subsystem – states

- From the beginning of core/neighbour.c:
- Is it a (latent) bug ?

```
if (!(state & NUD_IN_TIMER)) {
  #ifndef CONFIG_SMP
    printk(KERN_WARNING "neigh: timer & !nud_in_timer\n");
  #endif
goto out;
}
```

Neighboring Subsystem – states

- Special states:
- NUD_NOARP
- NUD_PERMANENT
- No state transitions are allowed from these states to another state.

Neighboring Subsystem – states

- NUD state combinations:
- NUD_IN_TIMER (NUD_INCOMPLETE|NUD_REACHABLE|NUD_DELAY|NUD_PROBE)
  - When removing a neighbour, we stop the timer (call *del_timer()*) only if the state is NUD_IN_TIMER.
- NUD_VALID (NUD_PERMANENT|NUD_NOARP|NUD_REACHABLE|NUD_PROBE|NUD_STALE|NUD_DELAY)
- NUD_CONNECTED (NUD_PERMANENT|NUD_NOARP|NUD_REACHABLE)

Neighbour states

- When a neighbour is in a STALE state it will remain in this state until one of the two will occur
  - a packet is sent to this neighbour.
  - Its state changes to FAILED.
- *neigh_resolve_output()* and *neigh_connected_output().*
- *net/core/neighbour.c*
- A neighbour in INCOMPLETE state does not have MAC address set yet (ha member of neighbour)
- So when *neigh_resolve_output()* is called, the neighbour state

  is changed to INCOMPLETE.

Neighbour states

- When *neigh_connected_output()* is called, the MAC address of the neighbour is known; so we end up with calling *dev_queue_xmit()*, which calls the *hard_start_xmit()* method of the NIC device driver.
- The *hard_start_xmit()* method actually puts the frame on the wire.

IPSec

- Works at network IP layer (L3)
- Used in many forms of secured networks like VPNs.
- Mandatory in IPv6. (not in IPv4)
- Implemented in many operating systems: Linux, Solaris, Windows, and more.
- In 2.6 kernel : implemented by Dave Miller and Alexey Kuznetsov.
- Transformation bundles.
- Chain of dst entries; only the last one is for routing.
- The dst entries in the chain have A NULL Neighbor as a member.
  - (except the last one)

IPSec-cont.

- RFC2401

IPSec-cont.

- User space tools: http://ipsec-tools.sf.net
- Building VPN : http://www.openswan.org/ (Open Source).
- There are also non IPSec solutions for VPN
  - OpenVPN uses ssl/tls.
  - example: pptp
- struct xfrm_policy has the following member:
  - struct dst_entry *bundles.
  - __xfrm4_bundle_create() creates dst_entries (with the DST_NOHASH flag) see: *net/ipv4/xfrm4_policy.c*
- Transport Mode and Tunnel Mode.

IPSec-contd.

- Show the security policies:
  - *ip xfrm policy show*
- Create RSA keys:
  - *ipsec rsasigkey --verbose 2048 > keys.txt*
  - *ipsec showhostkey --left   > left.publickey*
  - *ipsec showhostkey --right > right.publickey*

IPSec-contd.

Example: Host to Host VPN (using openswan)

in */etc/ipsec.conf:*

```
conn linux-to-linux
 left=192.168.0.189
 leftnexthop=%direct
 leftrsasigkey=0sAQPPQ...
 right=192.168.0.45
 rightnexthop=%direct
 rightrsasigkey=0sAQNwb...
 type=tunnel
  auto=start
```

IPSec-contd.

- *service ipsec start* (to start the service)
- *ipsec verify* – Check your system to see if IPsec got installed and started correctly.
- *ipsec auto –status*
  - *If you see "IPsec SA established" , this implies success.*
- Look for errors in */var/log/secure* (fedora core) or in kernel syslog

Tips for hacking

- Documentation/networking/ip-sysctl.txt: networking kernel tunabels
- Example of reading a hex address:
- iph->daddr == 0x0A00A8C0 or

means checking if the address is 192.168.0.10 (C0=192,A8=168, 00=0,0A=10).

- A BASH script for getting MAC address from IP address: (ipToHex.sh)

```
#!/bin/sh
IP_ADDR=$1
for I in $(echo ${IP_ADDR}| sed -e "s/\./ /g"); do
 printf '%02X' $I
done
echo
```

usage example: ./ipToHex.sh 192.168.0.1 => C0A80001

Tips for hacking - Contd.

- Disable ping reply:
- echo 1 >/proc/sys/net/ipv4/icmp_echo_ignore_all
- Disable arp: ***ip link set eth0 arp off*** (the NOARP flag will be set)
- Also ***ifconfig eth0 -arp*** has the same effect.
- How can you get the Path MTU to a destination (PMTU)?
  - Use tracepath (see man tracepath).
  - Tracepath is from iputils.

Tips for hacking - Contd.

- **inet_addr_type()** method: returns the address type; the input to this method is the IP address. The return value can be RTN_LOCAL, RTN_UNICAST, RTN_BROADCAST, RTN_MULTICAST etc. See: net/ipv4/fib_frontend.c

Tips for hacking - Contd.

- In case you want to send a packet from a user space application through a specified device without altering any routing tables:

```
struct ifreq interface;
strncpy(interface.ifr_ifrn.ifrn_name, "eth1",IFNAMSIZ);
if (setsockopt(s, SOL_SOCKET, SO_BINDTODEVICE, (char
    *)&interface, sizeof(interface)) < 0)
 {
 printf("error setting SO_BINDTODEVICE");
 exit(1);
 }
```

Tips for hacking - Contd.

- Keep iphdr struct handy (printout): (from linux/ip.h)

```
struct iphdr {

  __u8    ihl:4,
    version:4;
    __u8  tos;
    __be16      tot_len;
    __be16      id;
    __be16      frag_off;
    __u8  ttl;
    __u8  protocol;
    __sum16     check;
    __be32      saddr;
    __be32      daddr;
     /*The options start here. */

};
```

Tips for hacking - Contd.

- NIPQUAD() : macro for printing hex addresses
- Printing mac address (from net_device):

```
printk("sk_buff->dev =%02x:%02x:%02x:%02x:%02x:%02x\n",
((skb)->dev)->dev_addr[0], ((skb)->dev)->dev_addr[1],
((skb)->dev)->dev_addr[2],((skb)->dev)->dev_addr[3],
((skb)->dev)->dev_addr[4], ((skb)->dev)->dev_addr[5]);
```

- Printing IP address (primary_key) of a neighbour (in hex format):

```
printk("neigh->primary_key =%02x.%02x.%02x.%02x\n",
neigh->primary_key[0], neigh->primary_key[1],
neigh->primary_key[2],neigh->primary_key[3]);
```

Tips for hacking - Contd.

- Or:

  ```
  printk("***neigh->primary_key= %u.%u.%u.%u\n",
      NIPQUAD(*(u32*)neigh->primary_key));
  ```

- CONFIG_NET_DMA is for TCP/IP offload.
- When you encounter: xfrm / CONFIG_XFRM this has to to do with IPSEC. (transformers).

Tips for hacking - Contd.

- Showing arp statistics by:
- ***cat /proc/net/stat/arp_cache***

entries allocs destroys hash_grows lookups hits res_failed rcv_probes_mcast rcv_probes_ucast periodic_gc_runs forced_gc_runs

periodic_gc_runs: statistics of how many times the *neigh_periodic_timer()* is called.

Links and more info

1) Linux Network Stack Walkthrough (2.4.20):

http://gicl.cs.drexel.edu/people/sevy/network/Linux_network_stack_wa

2) Understanding the Linux Kernel, Second Edition

By Daniel P. Bovet, Marco Cesati

Second Edition December 2002

chapter 18: networking.

- Understanding Linux Network Internals, Christian benvenuti

Oreilly , First Edition.

Links and more info

3) Linux Device Driver, by Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman

   Third Edition February 2005.

   – Chapter 17, Network Drivers

4) Linux networking: (a lot of docs about specific networking topics)

   – http://linux-net.osdl.org/index.php/Main_Page

5) netdev mailing list: http://www.spinics.net/lists/netdev/

Links and more info

6) Removal of multipath routing cache from kernel code:

http://lists.openwall.net/netdev/2007/03/12/76
http://lwn.net/Articles/241465/

7) Linux Advanced Routing & Traffic Control :

http://lartc.org/

8) ebtables – a filtering tool for a bridging:

http://ebtables.sourceforge.net/

Links and more info

9) **Writing Network Device Driver for Linux**: (article)

   – http://app.linux.org.mt/article/writing-netdrivers?locale=en

Links and more info

10) Netconf – a yearly networking conference; first was in 2004.

- http://vger.kernel.org/netconf2004.html
- http://vger.kernel.org/netconf2005.html
- http://vger.kernel.org/netconf2006.html
- Next one: Linux Conf Australia, January 2008,Melbourne
- David S. Miller, James Morris , Rusty Russell , Jamal Hadi Salim ,Stephen Hemminger , Harald Welte, Hideaki YOSHIFUJI, Herbert Xu ,Thomas Graf ,Robert Olsson ,Arnaldo Carvalho de Melo and others

Links and more info

11) **Policy Routing With Linux** - Online Book Edition

- by Matthew G. Marsh (Sams).
- http://www.policyrouting.org/PolicyRoutingBook/

12) THRASH - A dynamic LC-trie and hash data structure:

Robert Olsson Stefan Nilsson, August 2006

http://www.csc.kth.se/~snilsson/public/papers/trash/trash.pdf

13) IPSec howto:

http://www.ipsec-howto.org/t1.html

Links and more info

14) Openswan: Building and Integrating Virtual Private Networks , by Paul Wouters, Ken Bantoft

http://www.packtpub.com/book/openswan/mid/061205jqdnh2by

publisher: Packt Publishing.

15) a book including chapters about LVS:

“The Linux Enterprise Cluster- Build a Highly Available Cluster with Commodity Hardware and Free Software”, By Karl Kopper.

http://www.nostarch.com/frameset.php?startat=cluster

15) http://www.vyatta.com - Open-Source Networking

Links and more info

16) Address Resolution Protocol (ARP)

   – http://linux-ip.net/html/ether-arp.html

17) ARPWatch – a tool for monitor incoming ARP traffic.

   Lawrence Berkeley National Laboratory -

   ftp://ftp.ee.lbl.gov/arpwatch.tar.gz.

18) arptables:

   http://ebtables.sourceforge.net/download.html

19) TCP/IP Illustrated, Volume 1: The Protocols

   By W. Richard Stevens

   http://www.informit.com/store/product.aspx?isbn=0201633469

Links and more info

20) Unix Network Programming, Volume 1: The Sockets Networking API (3rd Edition) (Addison-Wesley Professional Computing Series) (Hardcover)

by W. Richard Stevens (Author), Bill Fenner (Author), Andrew M. Rudoff (Author)

Questions

- Questions ?
- Thank You !